**TaskMaster – Inginerie Software**

Proiect realizat de Virtejeanu Larisa-Ioana, Mohammad Shahsavari,

Alperen Soydan și Mustatoiu Ioan-Sebastian

**Prezentare Generală**

TaskMaster este o aplicație web concepută pentru a ajuta utilizatorii să-și gestioneze eficient task-urile, să-și monitorizeze progresul și să primească notificări pentru termenele limită care se apropie. Permite utilizatorilor să adauge, să actualizeze și să șteargă task-uri, să stabilească priorități și să-și personalizeze experiența. Acest document prezintă viziunea proiectului, funcționalitățile, scenariile de utilizare, poveștile utilizatorilor și planul de dezvoltare.

**Viziunea produsului**

* **PENTRU:** Utilizatori care au nevoie de o modalitate organizată de a-și gestiona task-urile.
* **CARE**: Se confruntă cu provocări în organizarea, prioritizarea și monitorizarea task-urilor.
* **SISTEMUL TaskMaster**: O aplicație web pentru gestionarea task-urilor.
* **CARE**: Oferă o interfață prietenoasă pentru gestionarea task-urilor, setarea notificărilor și prioritizarea activităților.
* **SPRE DEOSEBIRE DE**: Listele simple de tip to-do sau trackerele generice de task-uri.
* **ACEST PRODUS**: Integrează gestionarea task-urilor cu notificări, personalizare pentru utilizatori și accesibilitate.

### Funcționalități

1. **Crearea și Gestionarea Task-urilor**: Adăugarea, editarea și ștergerea task-urilor, cu posibilitatea de a seta termene limită și priorități.
2. **Sistem de Notificări și Mementouri**: Trimite notificări înainte de termenele limită ale task-urilor.
3. **Profilul Utilizatorului și Setări**: Personalizarea preferințelor pentru notificări și vizualizarea task-urilor.
4. **Tablou de Bord pentru Task-uri**: O prezentare vizuală a task-urilor, cu indicatori pentru priorități și termene limită.

**User Scenarios**

* **Scenariul 1**: O profesionistă își categorizează task-urile în muncă și personal pentru a le gestiona separat.
* **Scenariul 2**: Un student se bazează pe notificări pentru a ține evidența temelor și examenelor.
* **Scenariul 3**: Un manager de proiect are nevoie de o imagine de ansamblu asupra tuturor task-urilor pentru a urmări progresul echipei.

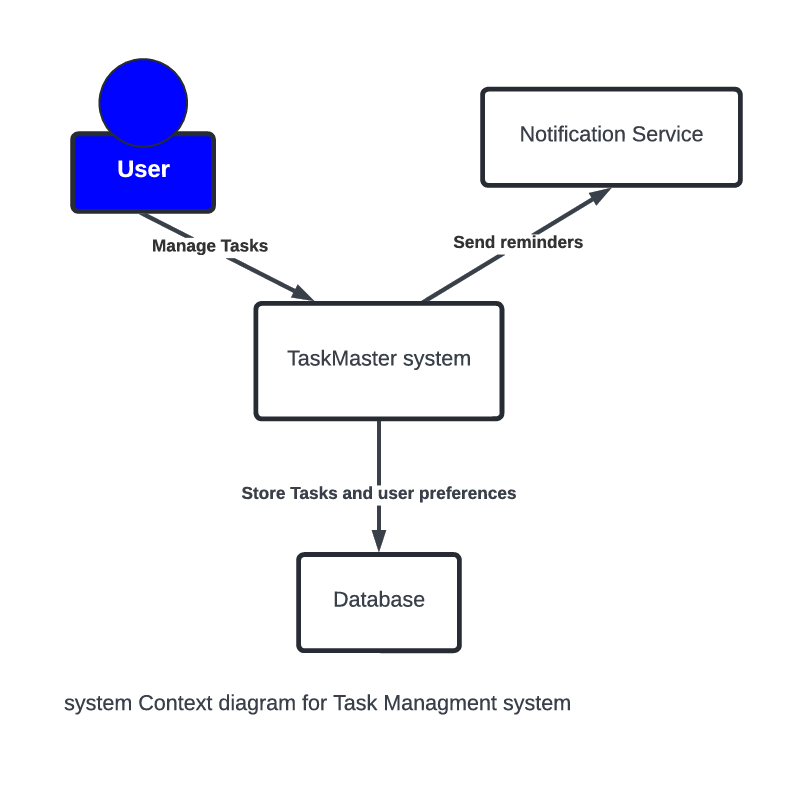
**User Stories**

* **Ca utilizator, vreau să adaug un nou task cu detalii (titlu, descriere, termen limită și prioritate) pentru a-mi gestiona eficient volumul de muncă.**
* **Ca utilizator, vreau să primesc notificări înainte de termenele limită ale task-urilor pentru a rămâne la zi.**
* **Ca utilizator, vreau să pot actualiza sau șterge task-uri pentru a-mi menține lista relevantă.**
* **Ca utilizator, vreau să văd toate task-urile într-un tablou de bord, cu prioritatea și termenele limită, pentru o urmărire ușoară.**

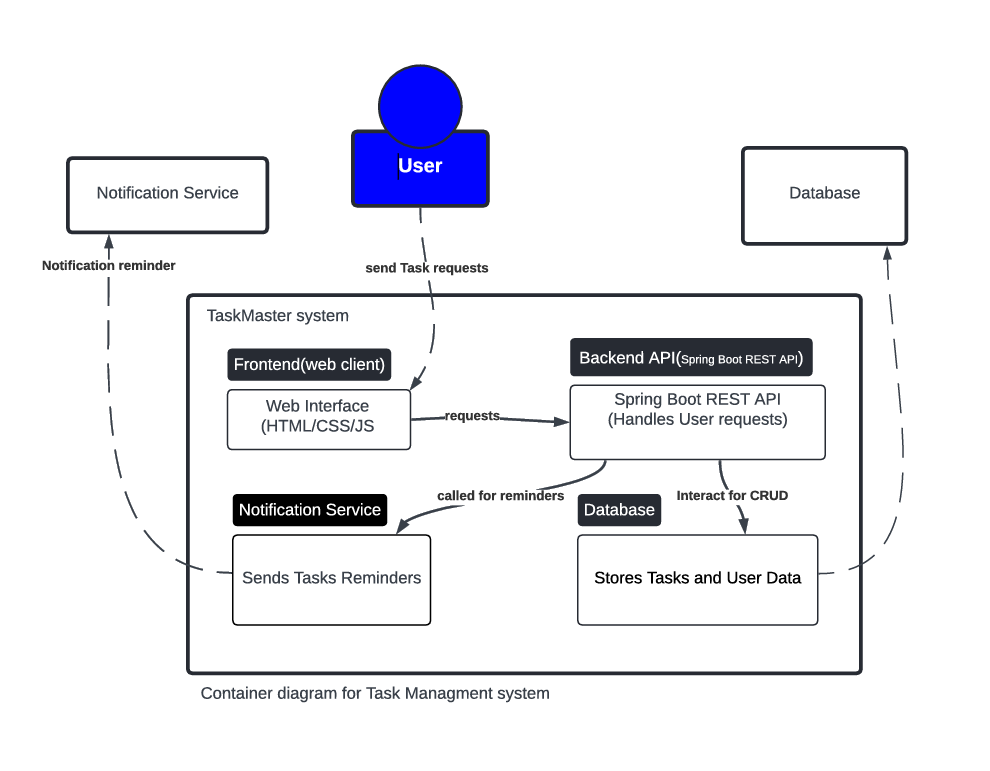
**Prioritizare backlog**

| Task | Descriere | Prioritate | Sprint |
| --- | --- | --- | --- |
| Configurarea Mediului de Dezvoltare | Configurarea repository-ului, a uneltelor de dezvoltare și a setărilor de proiect | Mare | 1 |
| Autentificarea Utilizatorilor | Crearea funcționalităților de logare și înregistrare | Mare | 1 |
| Sistem de Gestionare a Task-urilor | **Creare Task**: Permiterea utilizatorilor să creeze task-uri cu titlu, descriere, termen limită și prioritate | Mare | 1 |
|  | **Editare și Ștergere Task**: Posibilitatea de a edita sau șterge task-uri | Medie | 2 |
|  | **Tablou de Bord pentru Task-uri**: Afișarea task-urilor ordonate după prioritate și termen limită | Mare | 2 |
| Serviciu de Notificări | **Configurare Sistem de Mementouri**: Configurarea backend-ului pentru notificări | Mare | 3 |
| Testare și Corectare Erori | Efectuarea testelor și corectarea erorilor | Mare | Continuu |

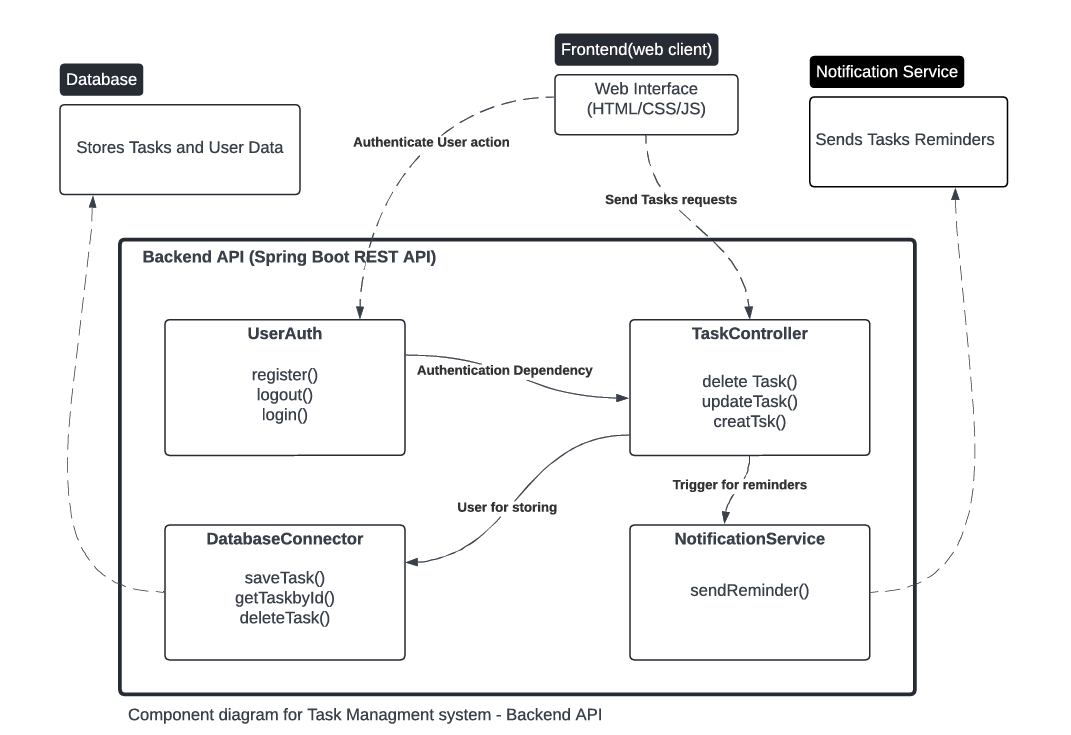
**Context Diagram**



**Container Diagrams**



**Component Diagrams**



Cerinte non-functionale și soluții pe care aplicația trebuie sa le îndeplinească pentru a asigura performanta, securitate și o experiența optima pentru utilizatori:

1. Performanta:

* Timp de răspuns: Sistemul trebuie sa răspundă la cererile înainte de către utilizatori în mai puțin de 2 secunde, atât pentru operațiuni de adaugare, editare și stergere de task-uri, cat și pentru afisarea listei de task-uri. Soluție: utilizarea SpringBoot pentru gestionarea eficienta a cererilor API și React + Vite pentru optimizarea vitezei de randare a interfetei. De asemenea, folosirea unor operațiuni asincrone și minimizarea interogărilor complexe în baza de date contribuie la reducerea timpului de răspuns.
* Scalabilitate: Arhitectura proiectului permite extinderea ușoară a acestuia, datorită separarii partilor de backend și frontend, ceea ce permite adaugarea de noi module fără impact major asupra aplicație existente. Arhitectura de tip REST API permite integrarea facila a noilor functionalitati și scalarea aplicației fără a afecta componentele deja existente.

2. Securitate:

* Autentificare și Inregistrare: Accesul la datele utilizatorului este protejat printr-un mecanism de autentificare. Fiecare utilizator poate accesa doar propriile task-uri. Autentificarea este gestionată printr-un sistem simplu, care stochează identificatorul utilizatorului în localStorage. Controlul accesului este realizat la nivel de backend, verificând ID-ul utilizatorului pentru fiecare cerere.
* Protecția datelor: Datele sensibile, cum ar fi informațiile de login și parolele, sunt gestionate în siguranța și nu sunt expuse în mod direct de către frontend sau API-uri.
* De asemenea, în baza de date, parolele sunt criptate.

3. Fiabilitate:

* Toleranta la erori: Sistemul gestioneaza corespunzător erorile de rețea și problemele de conectivitate la baza de date, oferind mesaje de eroare clare pentru utilizator. Implementarea unui mecanism de gestionare a erorilor la nivel de backend folosind **excepții personalizate** și returnarea de coduri de status HTTP corespunzătoare (ex: 404, 500). În frontend, sunt afişate mesaje de eroare intuitive pentru utilizatori.

4. Usability:

* Interfata prietenoasa: UI-ul este simplu și intuitiv, utilizand React pentru o experiența de utilizare dinamica și rapida. Folosirea de **component-based architecture** în React pentru a crea interfețe reutilizabile și optimizate, alături de o paletă de culori și elemente de design pentru o navigare intuitivă.

5. Notificari și automatizare:

* Serviciu de notificare prin e-mail: Implementarea unui sistem de notificare automata pentru deadline-urile task-urilor, utilizand JavaMailSender și un scheduler configurat în SpringBoot. Configurarea unui **scheduler** folosind **@Scheduled** din Spring pentru a verifica periodic deadline-urile task-urilor și trimiterea automată a notificărilor prin email folosind **JavaMailSender**.

**QA**

**1. Obiectivele testarii: Scopul principal al testării în cadrul aplicației TaskMaster este de a asigura funcționalitatea corectă a tuturor componentelor critice ale aplicației:**

* **Back-end (API-uri REST):** Testarea funcționalității pentru adăugarea, actualizarea, ștergerea și vizualizarea task-urilor.
* **Baza de date:** Verificarea conexiunii la baza de date și a integrității datelor stocate.
* **Funcționalități esențiale:** Testarea autentificării, înregistrării și sistemului de notificări prin e-mail.
* **Interfața utilizator:** Testare manuală pentru a verifica experiența utilizatorului și comportamentul aplicației în diverse scenarii.

**2. Procesul testarii: Testarea a fost realizată în diverse etape ale ciclului de viață al dezvoltării software (SDLC):**

* **Etapa de dezvoltare (Development):**
  + Testarea conexiunii la baza de date folosind scripturi Java (pentru a verifica stabilitatea conexiunii).
  + Testarea API-urilor prin Postman, pentru a valida funcționalitatea fiecărui endpoint.
* **Etapa de integrare (Integration):**
  + Verificarea interacțiunii dintre front-end și back-end pentru a ne asigura că datele sunt corect gestionate.
  + Testarea serviciului de trimitere email pentru notificările legate de deadline-uri.
* **Etapa de testare finală (Testing/QA):**
  + Testare manuală completă a aplicației, inclusiv pe partea de UI/UX.

**3.Metodele testarii:**

**Testare Manuală (Postman):**

* **Scop:** Validarea funcționalității API-urilor REST.
* **Ce am testat:**
  + **Crearea de task-uri:** Am verificat dacă datele introduse sunt salvate corect în baza de date.
  + **Actualizarea și ștergerea task-urilor:** Testarea corectitudinii celorlalte operații CRUD.
  + **Autentificarea și înregistrarea utilizatorilor:** Testarea procesului de login și restricționarea accesului la date.

**Testare Automată (Conexiunea la baza de date):**

* **Scop:** Validarea conexiunii cu baza de date MySQL.
* **Ce am testat:** Scriptul Java TestDBConnection verifică dacă aplicația se poate conecta la baza de date fără erori, asigurând stabilitatea infrastructurii.

**Testare Funcțională (Serviciul de Email):**

* **Scop:** Verificarea funcționării sistemului de notificări prin email.
* **Ce am testat:** Trimiterea automată a emailurilor de reminder pentru task-urile cu deadline apropiat.

**Testare de Integrare (Manuală):**

* **Scop:** Asigurarea integrării corecte dintre front-end și back-end.
* **Ce am testat:** Fluxul complet de adăugare, editare și ștergere a task-urilor prin interfața utilizator.

### ****4. Rezultatele Testării – Observații****

* **Testarea API-urilor:**
  + **Rezultat:** API-urile au răspuns corect la cereri GET, POST, PUT, DELETE.
  + **Observații:** Au fost identificate mici erori legate de validarea datelor (ex: lipsa unor mesaje de eroare clare), care au fost remediate ulterior.
* **Testarea bazei de date:**
  + **Rezultat:** Conexiunea a fost stabilă și fără erori.
  + **Observații:** Am ajustat unele setări pentru gestionarea mai eficientă a erorilor de conexiune.
* **Testarea sistemului de notificări prin email:**
  + **Rezultat:** Emailurile de reminder au fost trimise corect, conform programării.
* **Testarea UI/UX:**
  + **Rezultat:** Interfața este intuitivă, dar au existat observații privind alinierea și designul butoanelor, care au fost ajustate pe parcurs.

**Security Analysis**

**Principalele riscuri :**

Aplicația **TaskMaster** a fost dezvoltată având în vedere mai multe riscuri de securitate comune în aplicațiile web, iar pentru fiecare dintre acestea au fost implementate soluții specifice. Un risc important este expunerea datelor sensibile, cum ar fi parolele și informațiile de autentificare. Pentru a preveni acest lucru, parolele utilizatorilor sunt criptate utilizând algoritmul **BCrypt** înainte de a fi stocate în baza de date, ceea ce asigură că datele nu pot fi compromise chiar dacă baza de date ar fi accesată neautorizat. De asemenea, pentru protecția datelor în tranzit, este recomandată utilizarea protocolului **HTTPS**.

Un alt risc este reprezentat de atacurile de tip **SQL Injection**, care pot apărea dacă input-ul utilizatorilor este utilizat direct în interogări SQL. Pentru a preveni acest tip de atac, aplicația folosește **JPA și Hibernate**, care gestionează automat protecția împotriva SQL Injection prin utilizarea de interogări parametrizate. Acest lucru reduce semnificativ riscul de executare a unor comenzi SQL malițioase.

Pentru a proteja autentificarea utilizatorilor împotriva atacurilor de tip brute-force, aplicația validează parolele folosind funcția **passwordEncoder.matches()**, care compară parola introdusă cu cea criptată din baza de date. O îmbunătățire ar fi implementarea unui mecanism de limitare a numărului de încercări de autentificare nereușite și utilizarea de **token-uri JWT** pentru gestionarea sesiunilor, oferind un nivel suplimentar de securitate.

De asemenea, aplicația este vulnerabilă la atacuri de tip **Cross-Site Request Forgery (CSRF)**, deoarece protecția CSRF este dezactivată în prezent (csrf.disable()). În medii de producție, este recomandată activarea protecției CSRF și utilizarea de token-uri CSRF pentru cererile sensibile, cum ar fi cele de tip POST, PUT sau DELETE.

În ceea ce privește configurarea **CORS (Cross-Origin Resource Sharing)**, aplicația permite în prezent cereri doar de la http://localhost:5173, ceea ce este adecvat pentru dezvoltare locală. În producție, ar trebui restricționat accesul doar la domenii de încredere pentru a preveni accesul neautorizat la API-uri.

**CI/CD**

**Development (Dev):**

* + **Scop:** Mediul principal de dezvoltare unde sunt implementate și testate noile funcționalități de către dezvoltatori.
  + **Configurare:**
    - Server de backend local configurat pe **Spring Boot**.
    - Bază de date **MariaDB** rulând local prin **XAMPP**.
    - Frontend realizat cu **React + Vite**, rulând pe localhost:5173.
    - Testare API-uri realizată manual folosind **Postman**.
    - Configurări de securitate relaxate (CORS permis doar pentru localhost, CSRF dezactivat pentru testare).
  + **Avantaje:** Permite iterarea rapidă a codului și testarea locală a noilor funcționalități.

1. **Staging:**
   * **Scop:** Mediul de pre-producție unde sunt simulate condițiile din mediul de producție pentru a efectua testări mai riguroase.
   * **Configurare:**
     + Server de aplicație configurat similar cu mediul de producție.
     + Bază de date separată de cea de development, dar cu date de test relevante.
     + Configurări de securitate mai stricte, inclusiv activarea protecției CSRF și testarea limitărilor CORS.
   * **Avantaje:** Identificarea bug-urilor sau problemelor de performanță înainte de lansarea în producție.
2. **Production:**
   * **Scop:** Mediul final în care aplicația este disponibilă utilizatorilor finali.
   * **Configurare:**
     + Server dedicat configurat pentru performanță și securitate optimă.
     + Utilizarea unui server de baze de date securizat, cu backup automatizat.
     + Configurări stricte de securitate: HTTPS activ, protecție CSRF activată, autentificare avansată (posibil JWT).
     + Monitorizare activă pentru performanță și erori în timp real.
   * **Avantaje:** Stabilitate și securitate pentru utilizatori, cu un sistem robust de monitorizare.